# WEST Search History

DATE: Thursday, November 13, 2003

| Set Name side by side | Query | Hit Count | Set Name result set |
|---|---|---|---|
| *DB=USPT,PGPB; PLUR=NO; OP=ADJ* | | | |
| L1 | unpack instruction$1 | 67 | L1 |

END OF SEARCH HISTORY

WEST

Generate Collection Print

L1: Entry 30 of 67 File: USPT Jan 9, 2001

US-PAT-NO: 6173366
DOCUMENT-IDENTIFIER: US 6173366 B1
**** See image for Certificate of Correction ****

TITLE: Load and store instructions which perform unpacking and packing of data bits in separate vector and integer cache storage

DATE-ISSUED: January 9, 2001

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Thayer; John S. | Houston | TX | | |
| Favor; John G. | Scotts Valley | CA | | |
| Weber; Frederick D. | San Jose | CA | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| Compaq Computer Corp. | Houston | TX | | | 02 |
| Advanced Micro Devices, Inc. | Sunnyvale | CA | | | 02 |

APPL-NO: 08/ 759044 [PALM]
DATE FILED: December 2, 1996

INT-CL: [07] G06 F 12/04

US-CL-ISSUED: 711/129; 711/125, 712/4, 710/66, 710/68, 710/130
US-CL-CURRENT: 711/129; 710/66, 710/68, 711/125, 712/4

FIELD-OF-SEARCH: 711/118, 711/123, 711/125, 711/131, 711/220, 711/211, 711/217, 711/171, 711/173, 711/129, 345/202, 345/520, 345/521, 345/193, 345/198, 712/3, 712/4, 710/50, 710/66, 710/68, 710/130

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

| | PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|---|
| ☐ | 3541516 | November 1970 | Senzig | 714/6 |
| ☐ | 3701977 | October 1972 | Mendelson et al. | 711/126 |
| ☐ | 4783736 | November 1988 | Ziegler et al. | 711/130 |
| ☐ | 4884197 | November 1989 | Sachs et al. | 711/123 |
| ☐ | 4891754 | January 1990 | Boreland | |
| ☐ | 5025407 | June 1991 | Gulley et al. | |
| ☐ | 5193167 | March 1993 | Sites et al. | |
| ☐ | 5307300 | April 1994 | Komoto et al. | |
| ☐ | 5335330 | August 1994 | Inoue | 712/241 |
| ☐ | 5437043 | July 1995 | Fujii et al. | |
| ☐ | 5481713 | January 1996 | Wetmore et al. | 395/705 |
| ☐ | 5513366 | April 1996 | Agarwal et al. | 712/22 |
| ☐ | 5627981 | May 1997 | Adler et al. | |
| ☐ | 5640588 | June 1997 | Vegesna et al. | |
| ☐ | 5669013 | September 1997 | Watanabe et al. | |
| ☐ | 5801975 | September 1998 | Thayer et al. | |
| ☐ | 5845083 | December 1998 | Hamadani et al. | 709/231 |
| ☐ | 5893145 | April 1999 | Thayer et al. | |
| ☐ | 5909572 | June 1999 | Thayer et al. | |

OTHER PUBLICATIONS

Mahlke et al., "A Comparison of Full and Partial Prdicated Execution Support for ILP Processor" by 1995, IEEE Publication, pp. 138-149.
Kohn, L, et al., "The Visual Instruction Set (VIS) in UltraSPARC," SPARC Technology Business--Sun Microsystems, Inc., 1996 IEEE, pp. 462-489.
Gwennap, Linley, "UltraSparc Adds Multimedia Instructions--Other New Instructions Handle Unaligned and Little-Endian Data," Microprocessor Report, Dec. 5, 1994, pp. 16-18.
Lee, Ruby B., "Realtime MPEG Video via Software Decompression on a PA-RISC Processor," Hewlett-Packard Company, 1995 IEEE, pp. 186-192.
Mattison, Phillip E., "Practical Digital Video With Programming Examples in C," Wiley Professional Computing, pp. 158-178.
Zhou, Chang-Guo, et al., "MPEG Video Decoding With the UltraSPARC Visual Instruction Set," Sun Microsystems, Inc., 1995 IEEE, pp. 470-474.

ART-UNIT: 272

PRIMARY-EXAMINER: Peikari; B. James

ATTY-AGENT-FIRM: Conley, Rose & Tayon, P.C. Kowert; Robert C. Daffer; Kevin L.

ABSTRACT:

A multimedia extension unit (MEU) is provided for performing various multimedia-type operations. The MEU can be coupled either through a coprocessor bus or a local CPU bus to a conventional processor. The MEU employs vector registers, a vector ALU, and an operand routing unit (ORU) to perform a maximum number of the multimedia operations within as few instruction cycles as possible. Complex algorithms are readily performed by arranging operands upon the vector ALU in accordance with the desired algorithm flowgraph. The ORU aligns the operands within partitioned slots or sub-slots of the vector registers using vector instructions unique to the MEU. At the output of the ORU, operand pairs from vector source or destination registers can be easily routed

and combined at the vector ALU. The vector instructions employ special load/store instructions in combination with numerous operational instructions to carry out concurrent multimedia operations on the aligned operands.

31 Claims, 19 Drawing figures

**WEST**

Generate Collection | Print

L1: Entry 30 of 67 File: USPT Jan 9, 2001

DOCUMENT-IDENTIFIER: US 6173366 B1
**** See image for Certificate of Correction ****
TITLE: Load and store instructions which perform unpacking and packing of data bits in separate vector and integer cache storage

Brief Summary Text (41):
Arithmetic scaling which is lacking from many conventional operations is readily performed as part of the present load/store instructions. For example, packing and unpacking instructions found in many DSP instruction sets can be avoided. Thus, unpacking of an 8-bit word into a 20-bit slot occurs as part of a load instruction, whereas packing of a 20-bit operand to an 8-bit word occurs as part of the store instruction. Combining packing and unpacking operations into store and load helps eliminate unnecessary move operations which occur as part of stand-alone conventional pack and unpack instructions.

# WEST

Generate Collection | Print

L1: Entry 16 of 67 File: USPT Feb 4, 2003

US-PAT-NO: 6516406
DOCUMENT-IDENTIFIER: US 6516406 B1

TITLE: Processor executing unpack instruction to interleave data elements from two packed data

DATE-ISSUED: February 4, 2003

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Peleg; Alexander | Carmelia | | | IL |
| Yaari; Yaakov | Hanadin | | | IL |
| Mittal; Millind | Haifa | | | IL |
| Mennemeier; Larry M. | Boulder Creek | CA | | |
| Eitan; Benny | Haifa | | | IL |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| Intel Corporation | Santa Clara | CA | | | 02 |

APPL-NO: 09/ 657448 [PALM]
DATE FILED: September 8, 2000

PARENT-CASE:
RELATED APPLICATIONS Continuation of application Ser. No. 08/974,435, filed Nov. 20, 1997, now Pat. No. 6,119,216, which is a Divisional of Ser. No. 08/791,003, filed Jan. 27, 1997, now Pat. No. 5,802,336, which is a Continuation of Ser. No. 08/349,047, filed Dec. 2, 1994, abandoned.

INT-CL: [07] G06 F 9/315

US-CL-ISSUED: 712/225; 712/22, 712/223, 712/300
US-CL-CURRENT: 712/225; 712/22, 712/223, 712/300

FIELD-OF-SEARCH: 712/223, 712/225, 712/300, 712/22

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

| | PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|---|
| ☐ | 3711692 | January 1973 | Batcher | 708/210 |
| ☐ | 3723715 | March 1973 | Chen et al. | 708/709 |
| ☐ | 4139899 | February 1979 | Tulpule et al. | 712/224 |
| ☐ | 4161784 | July 1979 | Cushing et al. | 708/513 |
| ☐ | 4393468 | July 1983 | New | 708/518 |
| ☐ | 4418383 | November 1983 | Doyle et al. | 710/307 |
| ☐ | 4498177 | February 1985 | Larson | 714/806 |
| ☐ | 4707800 | November 1987 | Montrone et al. | 708/714 |
| ☐ | 4771379 | September 1988 | Ando et al. | 712/42 |
| ☐ | 4903228 | February 1990 | Gregoire et al. | 712/224 |
| ☐ | 4989168 | January 1991 | Kuroda et al. | 708/210 |
| ☐ | 5081698 | January 1992 | Kohn | 345/422 |
| ☐ | 5095457 | March 1992 | Jeong | 708/626 |
| ☐ | 5168571 | December 1992 | Hoover et al. | 712/210 |
| ☐ | 5187679 | February 1993 | Vassiliadis et al. | 708/706 |
| ☐ | 5268995 | December 1993 | Diefendorff et al. | 705/38 |
| ☐ | 5390135 | February 1995 | Lee et al. | 708/518 |
| ☐ | 5408670 | April 1995 | Davies | 712/16 |
| ☐ | 5423010 | June 1995 | Mizukami | 341/60 |
| ☐ | 5426783 | June 1995 | Norrie et al. | 712/225 |
| ☐ | 5465374 | November 1995 | Dinkjian et al. | 711/219 |
| ☐ | 5487159 | January 1996 | Byers et al. | 712/223 |
| ☐ | 5594437 | January 1997 | O'Malley | 341/67 |
| ☐ | 5625374 | April 1997 | Turkowski | 345/639 |
| ☐ | 5680161 | October 1997 | Lehman et al. | 345/531 |
| ☐ | 5781457 | July 1998 | Cohen et al. | 708/231 |
| ☐ | 5909552 | June 1999 | Jensen et al. | 709/234 |
| ☐ | 5938756 | August 1999 | Van Hook et al. | 712/23 |

FOREIGN PATENT DOCUMENTS

| FOREIGN-PAT-NO | PUBN-DATE | COUNTRY | US-CL |
|---|---|---|---|
| 0395348 | October 1990 | EP | |

OTHER PUBLICATIONS

Diefendorff, Keith, et al. "Organization of the Motorola 88110 Superscalar RISC Microprocessor", pp. 40-63, (12)Apr. 1992, No. 2, Los Alamitos CA IEEE Micro.
Kawakami, Y., et al., "A Single-Chip Digital Signal Processor for Voiceband Applications," IEEE, 1980 International Solid-State Circuits Conference, pp. 40-41.
UltraSPARC Multimedia Capabilities On-Chip Support for Real0-Time Video and Advanced Graphics; SPARC Technology Business, Sep. 1994, Sun Microsystems, Inc.
Case, B., "Philips Hopes to Displace DSPs with VLIW, TriMedia Processors Aimed at Future Multimedia Embedded Apps," Microprocessor Report, Dec. 1994, pp. 12-18.
Gwennap, L., "New PA-RISC Processor Decodes MPEG Video, H's PA-7100LC Uses New Instructions to Eliminate Decoder Chip," Microprocessor Report, Jan. 1994, pp. 16-17.
TMS320c2X, User's Guide, Digital Signal Processing Products, Texas Instruments, 1993,

pp. 3-2-3-11; 3-28-3-34;4-1-4-22;4-41;4-103; 4-119; 4-120; 4-122; 4-150; 4-151.
i860 TM. Microprocessor Family Programmer's Reference Manual, Intel Corporation, 1992, Chapters 1, 3, 8 and 12.
Lee, R.B., "Accelerating Multimedia with Enhanced Microprocessors," IEEE Micro, Apr. 1995, pp. 22-32.
Pentium Processor's User's Manual, vol. 3: Architecture and Programming Manual, Intel Corporation, 1993, Chapters 1, 3, 4, 6, 8, and 18.
Margulis, N., "i860 Microprocessor Architecture," McGraw Hill, Inc., 1990, Chapters 6, 7, 8, 10, and 11.
Intel i750, i860 TM, i960 Processors and Related Products, 1993, pp. 1-3.
Motorola MC88110 Second Generation RISC Microprocessor User's Manual, Motorola, Inc., 1991.
MC88110 Second Generation-RISC Microprocessor User's Manual, Motorola, Inc., Sep. 1992, pp. 2-1 through 2-22, 3-1 through 3-32, 5-1 through 5-25, 10-62 through 10-71, Index 1 through 17.
Errata to MC88110 Second Generation RISC Microprocessor User's Manual, Motorola, Inc., 1992, pp. 1-11.
MC88110 Programmer's Reference Guide, Motorola, Inc., 1992, pp. 1-4.
Shipnes,J., "Graphics Processing with the 88110 RISC Microprocessor," Motorola, Inc., IEEE, No. 0-8186-26455-0/92, 1992, pp. 169-174.

ART-UNIT: 2183

PRIMARY-EXAMINER: Kim; Kenneth S.

ATTY-AGENT-FIRM: Blakely, Sokoloff, Taylor & Zafman LLP

ABSTRACT:

An apparatus includes an instruction decoder, first and second source registers and a circuit coupled to the decoder to receive packed data from the source registers and to unpack the packed data responsive to an unpack instruction received by the decoder. A first packed data element and a third packed data element are received from the first source register. A second packed data element and a fourth packed data element are received from the second source register. The circuit copies the packed data elements into a destination register resulting with the second packed data element adjacent to the first packed data element, the third packed data element adjacent to the second packed data element, and the fourth packed data element adjacent to the third packed data element.

18 Claims, 18 Drawing figures

WEST

Generate Collection Print

L1: Entry 16 of 67 File: USPT Feb 4, 2003

DOCUMENT-IDENTIFIER: US 6516406 B1
TITLE: Processor executing unpack instruction to interleave data elements from two packed data

Abstract Text (1):
An apparatus includes an instruction decoder, first and second source registers and a circuit coupled to the decoder to receive packed data from the source registers and to unpack the packed data responsive to an unpack instruction received by the decoder. A first packed data element and a third packed data element are received from the first source register. A second packed data element and a fourth packed data element are received from the second source register. The circuit copies the packed data elements into a destination register resulting with the second packed data element adjacent to the first packed data element, the third packed data element adjacent to the second packed data element, and the fourth packed data element adjacent to the third packed data element.

CLAIMS:

1. An apparatus comprising: a instruction decoder to receive an unpack instruction; a first source register to hold a first packed data having a first plurality of packed data elements including a first packed data element and a third packed data element; a second source register to hold a second packed data having a second plurality of packed data elements including a second packed data element and a fourth packed data element; a destination register to hold a third packed data; a circuit coupled to the decoder to receive the first packed data from the first source register and the second packed data from the second source register and to unpack the first packed data and the second packed data responsive to the unpack instruction by copying the first packed data element into the destination register, copying the second packed data element into the destination register adjacent to the first packed data element, copying the third packed data element into the destination register adjacent to the second packed data element, and copying the fourth packed data element into the destination register adjacent to the third packed data element.

2. The apparatus of claim 1, the unpack instruction having an Intel integer opcode format comprising three bytes, a third byte of the three bytes permitting a source register address and a source-destination register address.

7. The apparatus of claim 2 wherein the decoder further decodes the unpack instruction, a first byte and a second byte of the three bytes comprising an operation code specifying an unpack operation to interleave low order packed elements from the first and second packed data, the elements selected from the group consisting of byte elements, word elements and doubleword elements.

8. The apparatus of claim 2 further comprising: a memory to hold the unpack instruction; and a storage device to hold software, the software configured to supply the unpack instruction to the memory for execution.

9. The apparatus of claim 8, the instruction decoder to receive and decode the unpack instruction from the memory, the first source register corresponding to the source register address, the second source register corresponding to the source-destination register address.

12. The apparatus of claim 1 wherein the first packed data element is a low order data

element of the first packed data and the second packed data element is a low order data element of the second packed data and the unpack instruction comprises an opcode field to contain one of a set of operation codes to specify an unpack operation interleaving low order data elements from the first and the second pluralities of packed data elements, the opcode field specifying data elements selected from the group consisting of byte elements, word elements and doubleword elements.

13. The apparatus of claim 12 wherein the opcode field of the unpack instruction contains one of a set of operation codes comprising the hexadecimal values 0F60, 0F61 and 0F62.

14. The apparatus of claim 1 wherein the first packed data element is a high order data element of the first packed data and the second packed data element is a high order data element of the second packed data and the unpack instruction comprises an opcode field to contain one of a set of operation codes to specify an unpack operation interleaving high order data elements from the first and the second pluralities of packed data elements, the opcode field specifying data elements selected from the group consisting of byte elements, word elements and doubleword elements.

15. The apparatus of claim 14 wherein the opcode field of the unpack instruction contains one of a set of operation codes comprising the hexadecimal values 0F68, 0F69 and 0F6A.

18. A computer system comprising: a memory to hold an unpack instruction having an Intel integer opcode format comprising three or more bytes, one of the three or more bytes permitting a first three-bit source register address and a second three-bit source-destination register address; a storage device to hold software, the software configured to supply the unpack instruction to the memory for execution; a processor enabled to receive and decode the unpack instruction from the memory, the processor including: a first register corresponding to the first three-bit source register address to hold a first packed data having a first plurality of packed data elements including a first packed data element and a third packed data element, a second register corresponding to the second three-bit source-destination register address to hold a second packed data having a second plurality of packed data elements including a second packed data element and a fourth packed data element, and a circuit to receive the first packed data from the first register and the second packed data from the second register and to copy the first packed data element into the second register, copy the second packed data element into the second register adjacent to the first packed data element, copy the third packed data element into the second register adjacent to the second packed data element, and copy the fourth packed data element into the second register adjacent to the third packed data element.

# WEST

Generate Collection Print

L1: Entry 60 of 67 File: USPT Oct 7, 1997

US-PAT-NO: 5675777
DOCUMENT-IDENTIFIER: US 5675777 A
**** See image for Certificate of Correction ****

TITLE: Architecture for minimal instruction set computing system

DATE-ISSUED: October 7, 1997

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Glickman; Jeff Bret | Champaign | IL | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| Hipercore, Inc. | Champaign | IL | | | 02 |

APPL-NO: 08/ 445973 [PALM]
DATE FILED: May 22, 1995

PARENT-CASE:
This application is a continuation of application Ser. No. 08/191,049, filed Feb. 1, 1994, now abandoned, which is a continuation of application Ser. No. 08/100,698, filed Jul. 30, 1993, and now abandoned, which is a continuation of application Ser. No. 07/965,524 of Oct. 23, 1992, now abandoned, which was a continuation of application Ser. No. 07/471,962 filed Jan. 29, 1990, which is now abandoned.

INT-CL: [06] G06 F 9/22, G06 F 9/315

US-CL-ISSUED: 395/561; 364/232.23, 364/258, 364/259, 364/259.8
US-CL-CURRENT: 712/220

FIELD-OF-SEARCH: 395/375, 395/800

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

| PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|
| 3700873 | October 1972 | Yhap | 364/200 |
| 4040031 | August 1977 | Cassonnet | 364/200 |
| 4231085 | October 1980 | Bazlen et al. | 364/200 |
| 4376976 | March 1983 | Lahti et al. | 364/200 |
| 4442488 | April 1984 | Hall | 364/200 |
| 4454580 | June 1984 | Page et al. | 364/200 |
| 4566063 | January 1986 | Zolnowsky et al. | 364/200 |
| 4569016 | February 1986 | Hao et al. | 364/200 |
| 4589065 | May 1986 | Auslander et al. | 364/200 |
| 4589087 | May 1986 | Auslander et al. | 364/200 |
| 4594659 | June 1986 | Guenthner et al. | |
| 4626988 | December 1986 | George | 364/200 |
| 4635194 | January 1987 | Burger et al. | 364/200 |
| 4714994 | December 1987 | Oklobdzija et al. | 364/200 |
| 4729093 | March 1988 | Mothersole et al. | 364/200 |
| 4734852 | March 1988 | Johnson et al. | |
| 4739471 | April 1988 | Baum et al. | |
| 4761731 | August 1988 | Webb | 364/200 |
| 4766566 | August 1988 | Chuang | 364/200 |
| 4803620 | February 1989 | Inagami et al. | 364/200 |
| 4851990 | July 1989 | Johnson et al. | |
| 4926323 | May 1990 | Baror et al. | |
| 5050068 | September 1991 | Dollas et al. | 395/375 |
| 5201056 | April 1993 | Daniel et al. | 395/800 |

ART-UNIT: 234

PRIMARY-EXAMINER: Teska; Kevin J.

ASSISTANT-EXAMINER: Mohamed; Ayni

ATTY-AGENT-FIRM: Brinks Hofer Gilson & Lione

ABSTRACT:

A computing system architecture that uses a minimal instruction set for the functioning of a general purpose computing system as well as applying completely unencoded instructions from memory directly to the hardware is herein described. The present invention additionally uses a flowthrough design to further reduce the hardware complexity to provide a streamlined and extremely efficient architecture.

8 Claims, 8 Drawing figures

WEST

Generate Collection Print

L1: Entry 60 of 67 File: USPT Oct 7, 1997

DOCUMENT-IDENTIFIER: US 5675777 A
**** See image for Certificate of Correction ****
TITLE: Architecture for minimal instruction set computing system

Detailed Description Text (82):
The preferred embodiment of the present invention supports ANSI/IEEE 754 floating point computations with the use of circuitry shown as functional blocks 68 and 66 to provide a sequence of unpack and pack operations. Floating point computations take place completely on the main chip of processor 10 to eliminate an external floating point coprocessor. However, floating point computational speed could be increased with the addition of a floating point coprocessor. Two atomic instructions, floating point unpack (UNPACK) and floating point pack (PACK), make this possible. The floating point unpack instruction takes two floating point numbers and separates each of the floating point numbers into an integer mantissa and an integer exponent which are stored in the floating point register file 82. This unpack operation may be performed in one of two modes: unpack for floating point addition or unpack for floating point multiplication.

Detailed Description Text (83):
These integer values can be manipulated by the adder/subtracter 60, multiplier 62, or logic/shift circuits 64. After all the operations have been performed on the mantissas and exponents, the floating point packer circuit 66 combines the mantissa and exponent to form a new floating point word. Accordingly, a floating point operation will require the sequence of an unpack instruction, a sequence of integer instructions, and a pack instruction.

Detailed Description Text (84):
The floating point unpacker circuit 68 performs the first operation for floating point multiplication, addition, or subtraction. The unpacker circuit 68 accepts two floating point operands at lines 27 and 29 and produces integer quantities that are manipulated with the adder/subtracter circuit 60, multiplier circuit 62, and logic/shift circuits 64, as described above. A control bit from the ICW 18 provides the unpacker circuit 68 with an unpack instruction via control line 30a for either addition or multiplication.

WEST

Generate Collection | Print

L1: Entry 63 of 67 File: USPT Jun 20, 1995

US-PAT-NO: 5426783
DOCUMENT-IDENTIFIER: US 5426783 A
**** See image for Certificate of Correction ****

TITLE: System for processing eight bytes or less by the move, pack and unpack instruction of the ESA/390 instruction set

DATE-ISSUED: June 20, 1995

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Norrie; Chris | San Jose | CA | | |
| Rawlinson; Stephen J. | Sunnyvale | CA | | |
| Zmyslowski; Allan | Sunnyvale | CA | | |

ASSIGNEE-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY | TYPE CODE |
|---|---|---|---|---|---|
| Amdahl Corporation | Sunnyvale | CA | | | 02 |

APPL-NO: 07/ 970418 [PALM]
DATE FILED: November 2, 1992

INT-CL: [06] G06 F 9/24, G06 F 9/30, G06 F 9/38

US-CL-ISSUED: 395/800; 364/229.5, 364/232.23, 364/240.5, 364/254.7, 364/255.7, 364/255.8, 364/262.4, 364/262.81, 364/262.9, 364/DIG.1, 364/DIG.2, 395/375
US-CL-CURRENT: 712/225

FIELD-OF-SEARCH: 395/375, 395/800, 395/400, 395/575, 395/425, 395/600, 395/650, 395/500, 395/550, 395/325, 395/775, 395/725, 364/DIG.1, 364/DIG.2

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

| PAT-NO | ISSUE-DATE | PATENTEE-NAME | US-CL |
|---|---|---|---|
| 4376976 | March 1983 | Lahti et al. | 395/375 |
| 4750112 | June 1988 | Jones et al. | 395/375 |
| 4926323 | May 1990 | Baror et al. | 395/375 |
| 5269017 | December 1993 | Hayden et al. | 395/575 |

ART-UNIT: 232

PRIMARY-EXAMINER: Bowler; Alyssa H.

ASSISTANT-EXAMINER: Pan; Daniel H.

ATTY-AGENT-FIRM: Fliesler, Dubb, Meyer & Lovejoy

ABSTRACT:

A processing system comprising a first means for generating first signals indicating when the next instruction can begin processing where eight or less bytes are processed by the MOVE, PACK or UNPACK instruction, a second means for generating second signals if an overlap condition exists for the MOVE, PACK or UNPACK instruction being processed, and where the first means generates the first signals prior to the second means generating the second signals and independent of whether the second means generates the second signals.

10 Claims, 7 Drawing figures

WEST

Generate Collection | Print

L1: Entry 63 of 67 File: USPT Jun 20, 1995

DOCUMENT-IDENTIFIER: US 5426783 A
**** See image for Certificate of Correction ****
TITLE: System for processing eight bytes or less by the move, pack and unpack instruction of the ESA/390 instruction set

Abstract Text (1):
A processing system comprising a first means for generating first signals indicating when the next instruction can begin processing where eight or less bytes are processed by the MOVE, PACK or UNPACK instruction, a second means for generating second signals if an overlap condition exists for the MOVE, PACK or UNPACK instruction being processed, and where the first means generates the first signals prior to the second means generating the second signals and independent of whether the second means generates the second signals.

Brief Summary Text (4):
The IBM ESA/390 Principles of Operation defines the MOVE, PACK and UNPACK instructions.

Brief Summary Text (8):
In the UNPACK instruction the format of the second operand is changed from packed to zoned and the result is placed at the first operand address. The result is obtained as if the operands were processed right to left. When necessary, the second operand is considered to be extended on the left with zeros. If the first operand field is too short to contain all digits of the second operand, the remaining left-most portion of the second operand is ignored. Access exceptions for the unused portion of the second operand may or may not be indicated. When the operands overlap the result is obtained as if the operands were processed one byte at a time and as if the resulting bytes were stored immediately after fetching the first operand byte. The entire right-most second operand byte is used in forming the first result byte. For the remainder of the field information for the two result bytes is obtained from a single second operand byte and execution proceeds as if the left-most four bits of the byte were to remain available for the next result byte and need not be refetched. Thus, the result is as if the two result bytes were to be stored immediately after fetching a single operand byte. A field that is to be unpacked can be destroyed by improper overlapping. To save storage space for unpacking by overlapping the operands, the right-most byte of the first operand must be to the right of the right-most byte of the second operand by the number of bytes in the second operand minus two. If only one or two bytes are to be unpacked, the right-most byte of the two operands may coincide.

Brief Summary Text (9):
In each of these instructions bytes are fetched from operand 2 and stored in operand 1. When no overlap exists these instructions parallel process eight bytes of data at a time. When overlap does occur, in order to assure proper results, the instructions process the data effectively one byte at a time which is referred to as a one byte mode of operation. Many computer systems, invoking the use of the instruction set as set forth in International Business Machine Corporation's ESA/390 Principles of Operation, employ the concept of pipeline processing of instructions. In order to carry out a computer instruction a series of operations are performed by the computing system for each of the instructions within the instruction set. In a computer system using the pipeline concept of instruction processing, the instructions are broken down into a series of FLOWs where each FLOW contains a series of cycles. Referring to FIG. 2, the FLOWs for instructions are illustrated. Each FLOW is broken down into six cycles, a decode operation code cycle D, an address presentation cycle A, a

translation cycle T, a buffer access cycle B, an execution cycle X and, finally, a write or store cycle W. In order to process the instructions faster the FLOWs overlap such that different steps in the instructions are being processed at the same time rather than sequentially where each step is completed before starting the next step. In the MOVE, PACK and UNPACK instructions an overlap condition can exist between the location from which data can is being fetched from and the location in which the data, once processed, is stored. In many computer systems employing the pipeline system, eight bytes of data are parallel processed at the same time. When an overlap occurs, the requirements are that the instructions be carried out one byte at a time to generate the desired results. While overlap of operands within the MOVE, PACK and UNPACK instructions are the exception rather than the rule, the possibility of an overlap condition requires that overlap be tested for during each MOVE, PACK or UNPACK instruction such that the system can go from the mode of processing eight bytes at a time to the mode of processing one byte at a time.

Brief Summary Text (10):
Testing for an overlap condition in the MOVE, PACK and UNPACK instructions cannot be done until the instruction has obtained both the address from which data is to be fetched from and the address in which processed data is to be stored into. In a pipeline system the address of the location from which data is going to be fetched is available during cycle A of FLOW 1, the address in which data is going to be stored is available in cycle A of FLOW 2 and the determination of an overlap is performed during cycle T of FLOW 2. Cycle D in a FLOW is used to inform the system that the present FLOW is the last FLOW for this instruction and that the next instruction may be initiated in the next FLOW. The first cycle D to occur at the same time or after the occurrence of the cycle T in FLOW 2 is in FLOW 4. Thus, for those instructions requiring a testing for overlap, those instructions will use a minimum of four FLOWs to be completed regardless of whether the instruction could have been completed in less FLOWs if no overlap is encountered.

Brief Summary Text (11):
In processing the MOVE, PACK and UNPACK instructions the computing system processes the instructions with the assumption that an overlap condition does exist until it is determined otherwise in the T cycle of the FLOW 2. In the situations where eight bytes or less of data is to be processed by either the MOVE, PACK or UNPACK instruction and an overlap condition did not exist and, if it was not to require to test the instruction for an overlap condition, the MOVE instruction could be completed in two FLOWs and the PACK and UNPACK instructions could be completed within three FLOWs.

Brief Summary Text (13):
It is an object of the invention to execute the MOVE, PACK and UNPACK instructions without incurring the delay associated with determining whether an overlap condition exists.

Brief Summary Text (16):
It is another object of the invention to execute the UNPACK instruction in three FLOWs if both the number of bytes to be unpacked and the field length of the field to receive the unpacked data are both eight bytes or fewer and no overlap.

Brief Summary Text (17):
Briefly, the invention comprises means for determining if eight bytes or less are to be processed by the MOVE, PACK and UNPACK instructions and if so determined to signal that the instruction is in its last FLOW before the determination for the overlap condition is completed. Where thereafter an overlap condition for the MOVE instruction is detected, specific hardware is invoked for generating the proper resulting bytes as if the MOVE instruction had moved one byte of data at a time. The resulting bytes are made available for storage during the storage cycle of the last FLOW. In the PACK and UNPACK instructions, if an overlap condition is thereinafter detected, means are provided for trapping the PACK and UNPACK instruction thereby halting the processing of the instruction in the pipeline and branching to macrocode for the completion of the processing of the instruction. After the macrocode has completed processing the PACK or UNPACK instruction, the pipeline will be re-initiated for the next instruction following the instruction giving rise to the branch to macrocode.

Brief Summary Text (18):

The advantage of the present invention is to allow the PACK, UNPACK and MOVE instructions to be executed within the minimum number of FLOWs where an overlap condition is not detected thereby increasing the overall performance of the system when executing the MOVE, PACK or UNPACK instructions.

Drawing Description Text (4):
FIG. 2 depicts the FLOWs and cycles within each FLOW for the MOVE, PACK and UNPACK instructions.

Drawing Description Text (9):
FIG. 7 is a logic diagram for the PACK and UNPACK instructions for determining overlap and if a branch to macrocode should be taken and for generating a signal that the instruction will be completed at the end of three FLOWs.

Detailed Description Text (2):
The invention provides a system which avoids the necessary delay in determining if an overlap condition exists in a MOVE, PACK and UNPACK instruction when in fact no overlap condition exists. Each instruction is processed by a series of FLOWs where each FLOW contains a plurality of cycles. The FLOWs overlap each other such that multiprocessing can be taking place thereby allowing the instruction to be completed in the shortest period of time. A problem arises in such a pipeline when a given FLOW within the sequence of FLOWs requires information that is generated by a previous FLOW where the previous FLOW does not yet have the information available for the present FLOW.

Detailed Description Text (3):
Referring to FIG. 2, a series of FLOWs for the MOVE and PACK/UNPACK instructions are shown. Each FLOW as previously discussed comprises six cycles. In the MOVE instruction it is not until the T cycle of FLOW 2 that a determination can be made that an overlap condition exists. A determination can be made in the FLOW 1 of the MOVE instruction if the MOVE instruction is to move eight bytes or less. If eight bytes or less are to be moved by the MOVE instruction and no overlap condition exists, the MOVE instruction could not be completed by the end of FLOW 2. When an instruction is to be completed at the end of the FLOW 2 it is necessary to inform the system prior to the start of FLOW 3 that the system is to initiate the processing of the next instruction in FLOW 3. This invention generates a signal that the MOVE instruction can be completed at the end of FLOW 2 whenever the number of bytes to be moved is eight bytes or less, regardless of a possible overlap. When an overlap condition is sensed in the T cycle of the second FLOW, logic is provided to generate the proper resulting bytes for the instruction such that the instruction may be completed at the end of FLOW 2. Therefore, this invention allows the MOVE instruction to be completed in two FLOWs regardless of an overlap condition where the number of bytes to be moved are eight bytes or less. This results in a substantial saving of time in processing the MOVE instruction and enhances the performance of the overall computer system.

Detailed Description Text (4):
The PACK instruction would be completed in three FLOWs if the number of bytes to be processed for the PACK instruction is eight bytes or less. The UNPACK instruction would be completed in three FLOWs if the number of bytes to be processed is eight bytes or less and the field length for storing the resulting bytes is eight bytes or less. Again, the PACK and UNPACK instructions, if it is going to be completed in three FLOWs, must initiate a signal to the processor that such is the case. It is in the D cycle of a FLOW that the decision is made whether or not a new instruction can be started in the D cycle of the next FLOW. The invention generates in D cycle of FLOW 3 of the PACK and UNPACK instructions the signal that a new instruction should begin in FLOW 4 whenever the condition exists that the PACK and UNPACK instructions could have been completed within the third FLOW as described above. When in the T cycle of FLOW 2 a determination is made that an overlap condition exists and the PACK or UNPACK instruction would have been completed in three FLOWs if no overlap is detected, then the system branches to a macrocode program which emulates the instruction being processed and completes the instruction being processed. The pipeline is interrupted when such a branch occurs such that any FLOWs that are in process are completed but the effects of the processing of those FLOWs are nullified. When the macrocode has completed the instruction the macrocode returns back to the next instruction to be processed. The emulation of the PACK or UNPACK instruction by macrocode is well

understood by those skilled in the art and does not constitute part of this invention. Since the occurrence of an overlap situation in the PACK and UNPACK instruction is rare, the time saved by reducing the number of FLOWs necessary to complete the instruction results in enhancing the overall performance of the system.

Detailed Description Text (5):
Referring to FIG. 1, the instruction is received and stored in register 10. The operation code is in field OP code. Fields B1 and D1 are used to generate the address for the first operand OP1 and fields B2 and D2 are used to generate the address for the second operand OP2. In the MOVE instruction, fields L1 and L2 are combined to form one field L which defines the number of bytes to be moved during the MOVE instruction. In the PACK and UNPACK instructions, field L1 defines the length of the field in bytes in which data is to be stored and field L2 defines the number of bytes to be packed or unpacked. The number stored within fields L, L1 and L2 is one less than the number of actual bytes associated with that field.

Detailed Description Text (6):
Decoder 20 is connected to register 10 and receives from register 10 the contents of register 10. Decoder 20 will decode the OP code field and generate a MOVE signal on line 21 if the instruction decoded is a MOVE instruction, a PACK signal on line 22 if the instruction decoded is a PACK instruction, and an UNPACK signal on line 23 if the instruction decoded is an UNPACK instruction. Fields L1 and L2 are not processed by the decoder and are shown herein as being outputted on lines 24 and 25. Decoder 20 generates an address OP2 from fields B2 and D2 of register 10 and stores address OP2 in address register 30. Decoder 20 generates an address OP1 from fields B1 and D1 of register 10 and provides address OP1 by line 27 to be stored in address register 40. Addresses OP1 and OP2 are both 31 bits in length. The complement of address OP1 is provided to complement adder 50 by lines 41 and the address OP2 is provided on lines 31 to complement adder 50. Complement adder 50 adds the numbers generated on lines 41 and 31 and produces a series of sum bits 1 through 31 with the carry out bit being discarded. In effect, the one's complement adder 50 performs a subtraction between address OP2 and OP1 indicating the offset between address OP2 and OP1. Decoder 70 deciphers sum bits 1 through 25 and determines if all sum bits 1 through 25 are ones. Sum bits 1 through 25 will be all ones whenever address OP1 is greater than or equal to address OP2 by up to 63 bytes. The output of decoder 70 is provided on line 71 to overlap analysis logic 90. Decoder 80 deciphers sum bits 1 through 25 and determines if all sum bits 1 through 25 are zeros. Sum bits 1 through 25 will all be zeros whenever address OP2 is greater than address OP1 by up to 64 bytes. The output of decoder 80 is provided on line 81 to overlap analysis logic 90. Sum bits 26 through 31 are provided on lines 52 to 57 to overlap analysis logic 90.

Detailed Description Text (8):
Overlap analysis logic 90 interrogates its various inputs and generates (1) a MOVE overlap signal on line 91 whenever the MOVE instruction would move eight bytes or less and an overlap condition is detected, (2) a PACK overlap signal on line 92 whenever a PACK overlap is detected, (3) a PACK BRANCH TO MACROCODE signal on line 94 whenever a PACK overlap is detected and eight bytes or less are to be packed, (4) an UNPACK overlap signal on line 93 whenever an overlap is detected in an UNPACK instruction and (5) an UNPACK BRANCH TO MACROCODE signal on line 95 whenever eight bytes or less are to be unpacked and the length of the field to receive the UNPACK data is eight bytes or less.

Detailed Description Text (9):
Referring to FIG. 3, the inhibit overlap logic is shown in detail. AND 62 will be conditioned when the system is in the access register mode which will be true during a MOVE, PACK and UNPACK instruction as indicated on line 64, when the base registers in fields B1 and B2 of register 10 are not equal as indicated on line 65 and when the contents of the access register associated with the base register in field B1 is not equal to the contents of the access register associated with the base register field B2 as provided on line 66. When these three conditions are met the addresses OP2 and OP1 will be located in different address spaces. The state of AND 62, provided on line 67, is read into latch 63 by the system clock on line 68. Latch 63 provides an inhibit overlap signal on line 61 to the system.

Detailed Description Text (10):

In the MOVE, PACK and UNPACK instructions the data to be moved, packed or unpacked is located by address OP2 and the MOVE, UNPACK or PACK data is stored into the address indicated by OP1. Therefore, these instructions fetch the data from address OP2 and store the resulting data in address OP1.

Detailed Description Text (23):
In the PACK and UNPACK instructions both operands OP1 and OP2 have associated byte field lengths L1 and L2. The maximum number of bytes that can be PACKED or UNPACKED in a given instruction is 16. In the UNPACK instruction an overlap condition will be detected when address OP1 is equal to or greater than address OP2 and address OP2 plus field L2 is greater than or equal to address OP1. An overlap can also occur where address OP2 is greater than address OP1 and address OP1 plus field L1 is equal to or greater than address OP2.

Detailed Description Text (24):
More specifically, an overlap in the UNPACK instruction of only one byte will not be destructive nor will the occasion of fields L1 and L2 both being equal to zero give rise to a destructive result. Therefore, these conditions can be disregarded and only the remaining conditions need be tested for.

Detailed Description Text (26):
Referring to FIG. 7, field L1 is stored in register 510 and field L2 is stored in register 511. The contents of field L1 are referred to as A1 and the contents of field L2 are referred to as A2. Sum bits 28 through 31 are received via 1 ines 54 through 57 by register 512. Sum bits 28 through 31 are indicated by the symbol S and is provided to the rest of the logic via line 507. A binary one is provided on line 508 to be used by the logic. Decoder 515 produces an output when the contents A1 of register 510 are less than eight. Detector 516 interrogates the contents A2 of register 512 and produces an output when A2 is less than eight. An output from detector 516 on line 561 means that less than eight bytes of data is to be fetched by the instruction. The output of detector 515 on line 560 means that the length of the field to receive the data is less than eight bytes long. As previously stated, in an UNPACK instruction where there are less than eight bytes to be fetched and the field length of the location to which the data is to be stored is eight or less bytes then the instruction can be completed in three FLOWs. AND 533 receives the outputs of decoders 516 and 515 as well as for the UNPACK instruction signal on line 23. The coincidence of these three signals will condition AND 533 to produce an output signal 578 which will allow the UNPACK instruction during the D cycle of the third FLOW to inform the system to begin the operation of the next instruction at the beginning of the next FLOW. Again it should be noted that this condition is raised regardless of whether or not an overlap is detected by the remaining logic.

Detailed Description Text (27):
Condition 1a is tested by line 71 the output of decoder 70 and by line 590 which represents bits 26, 27 and 28 on lines 52, 53 and 53 from complement adder 50, the combination indicating that sum bits 1 through 28 are all ones. Condition 1b is tested by line 81 from decoder 80 and by line 591 which represents the inverted value of sum bits 26, 27 and 28 on lines 26, 27 and 28 from complement adder 50, the combination indicating that sum bits 1 through 28 are all zeros. Detector 514 provides a signal on line 562 whenever the contents A2 of register 511 are not equal to zero. Detector 513 will provide an output signal on line 563 whenever the contents A1 of register 510 is not equal to zero. AND circuit 521 is conditioned by the coincidence of signals appearing on lines 562 and 563 and provides an output on line 580. AND circuit 521 is conditioned when the criteria of 2a and 3a are met in the first set of conditions and conditions 2b and 3b are met in the second set of conditions. Adder 520 tests for the condition 4a. Adder 521 tests for the condition 4b. AND 523 will be conditioned upon the coincidence of all conditions la through 4a. AND 522 will be conditioned upon the coincidence of the conditions being met of 1b through 4b. The output of AND 522 and AND 523 are connected to OR 527 by lines 570 and 571 respectively. The output of OR 527 is connected to AND 529 by line 575. The INHIBIT OVERLAP signal is provided by line 61 to the negative input of AND 529 which will provide an UNPACK OVERLAP signal on line 93 whenever the inhibit overlap latch is not set, an UNPACK instruction is being processed and an overlap condition is detected by line 575. AND 531 will be conditioned when an UNPACK overlap condition has been sensed by AND 529, the number of bits to be unpacked is less than eight as indicated by the output of decoder 516 on

line 561, and the length of the field to which the unpacked data is to be stored is less than eight bytes as detected by detector 515. AND 531 will generate an UNPACK BRANCH TO MACROCODE signal on line 95. This signal will effectively cause the pipeline to recognize the UNPACK instruction will be performed by macrocode. The system will cause all unfinished FLOWs to be voided. Upon the macrocode completing the instruction, the macrocode will return back to the next instruction in the pipeline.

Detailed Description Text (31):
Referring to FIG. 7, the output of decoder 516 indicating that eight or less bytes are to be fetched in the instruction signal on line 561 is connected to AND·534. AND 534 is also conditioned by the PACK signal on line 22. When AND 534 is conditioned a signal is generated on line 579 indicating that the PACK instruction will be completed in FLOW 3 and allows the system to begin processing the next instruction in FLOW 4. Again, condition 1a is tested by line 71 from decoder 70 and conditions 1b and 1c are tested by line 81 from decoder 80. Condition 2a is tested by adder 517 and condition 2b is tested by adder 518. Adder 519 tests for conditions 3a and 2c. Adder 521 tests for conditions 3b and 3c. AND 524 tests for conditions 1a, 2a and 3a and provides an output when those conditions are met on line 572. AND 525 tests for conditions 1b, 2b and 3b and provides an output on line 573 when all these conditions are met. AND 526 is conditioned by the coincidence of conditions 1c, 2c and 3c and provides an output on line 574. OR 528 provides an output whenever either AND 524, 525 or 526 is conditioned. AND 530 will be conditioned when a condition output is sensed on line 576 from OR 528, a PACK instruction is being processed and the inhibit overlap latch is not set. The output of AND 530 is a PACK OVERLAP signal 92 indicating that a PACK overlap condition exists. AND 532 detects the condition that less than eight bytes or less are to be processed by the PACK instruction from the output of detector 516 on line 561 and the PACK OVERLAP signal on line 92 to generate a PACK BRANCH TO MACROCODE signal on line 94. As in the UNPACK instruction, the PACK BRANCH TO MACROCODE will cause the system to emulate the PACK instruction by macrocode and will allow those FLOWs in process to be voided. Finally, when the PACK macrocode is complete the system will re-initiate the pipeline with the next instruction to be processed.

Detailed Description Text (32):
It is readily apparent that both the PACK and UNPACK instructions for less than eight bytes and no overlap will be completed in three FLOWs thereby enhancing the overall performance of the system.

CLAIMS:

5. A processing system in a computer system for processing an UNPACK instruction within an instruction set for said computer system, said computer system employing pipeline architecture in the processing of said instruction set, said system comprising:

first means for generating an UNPACK THREE FLOW signal when eight or less bytes are to be unpacked by an UNPACK instruction presently being processed by said computer system and the length of a field in said computer system which is to receive the results of said UNPACK instruction presently being processed is eight bytes or less in length, said UNPACK THREE FLOW signal indicating that said UNPACK instruction needed only three FLOWs for processing and that a next instruction can begin to be processed during a next flow in the pipeline by said computer system following said UNPACK instruction flow presently being processed by said computer system;

second means for generating a second signal if an overlap condition exists for said UNPACK instruction being processed during said flow that said UNPACK THREE FLOW signal was generated by said first means: and

said first means generaling said UNPACK THREE FLOW signal prior to said second means generating said second signal and independent of whether said second means generates said second signal for minimizing the number of flows required to process eight bytes or less by said UNPACK instructions.

6. The processing system of claim 5 wherein said second means generates an UNPACK OVERLAP signal whenever an overlap condition exists between the fields associated with the processing of said UNPACK instruction.

8. A processing system in a computer system for processing a PACK instruction within an instruction set for said computer system, said computer system employing pipeline architecture in the processing of said instruction set, said system comprising:

first means for generating a PACK THREE FLOW signal when eight or less bytes are to be packed, said PACK THREE FLOW signal indicating that said PACK instruction needed only three FLOWs for processing and that a next instruction can begin to be processed during a next flow in the pipeline by said computer system following a said MOVE, PACK or UNPACK instruction flow presently being processed by said computer system:

second means for generating at second signal if an overlap condition exists for said PACK instruction being processed during said flow that said PACK THREE FLOW signal was generated by said first means; and

said first means generating said PACK THREE FLOW signal prior to said second means generating said second signal and independent of whether said second means generates said second signal for minimizing the number of flows required to process eight bytes or less by said PACK instruction.